Advanced Engineering Forum *Online: 2012-09-26*
ISSN: 2234-991X, Vols. 6-7, pp 1026-1030
doi:10.4028/www.scientific.net/AEF.6-7.1026
© 2012 The Author(s). Published by Trans Tech Publications Ltd, Switzerland.

Seamless Rendering of Large Scale Terrain

Baosong Deng[1, a], Tieqing Deng[2], Ronghuan Yu[2, b] and Jiawei Yu[1]

[1] Institute of logistics science, Beijing 100071, China

[2] Science and technology on information systems engineering laboratory, National University of defense technology, Changsha 410073, China

[a]dbs310@163.com, [b]yrh1983@163.com

Keywords: Large scale, Multi-resolution, Terrain rendering, GPU acceleration

Abstract. Terrain rendering has long been an active research topic in computer graphic and virtual reality. If large and detailed, digital terrains can be represented by a huge amount of data and therefore of graphical primitives to render in real-time. A dynamic, realistic and seamless rendering scheme for large scale terrain was proposed in this paper, based on successive LOD tiles and GPU acceleration. Multi-resolution girds and images were used for view-dependent data control and grid simplification, and multi-thread mechanism was employed for visibility clipping and data exchange between memory and disk, at the same time, a seamless combination algorithm between tiles of terrain and texture was proposed. Experimental results of real scenes with open data and comparisons with traditional method demonstrate the efficiency and practicality of our method.

Introduction

Visualization of landscapes and outdoor environments is important for various graphics applications, such as computer games and simulators [1-2]. Height field and image datasets are vital components of these applications. The advances in satellite imaging and cartography technologies have led to the generation of large terrain datasets that contain high resolution of objective areas[3]. Such terrains usually exceed the rendering capability of currently available graphics hardware, and thus reducing their complexity is mandatory for interactivity. Adjusting the complexity of terrain in memory according to view parameters is a common approach for interactive terrain rendering. Most LOD methods used a fixed representation approach [2]. With these methods, multiple representations of parts of terrain, typically square blocks, are pre-computed and stored off-line. At run-time, the appropriate approximation mesh is assembled from blocks based on the current view-parameters[4].

In this paper, we present a novel approach for interactive rendering of large terrain datasets, which is designed to prevent limitations of previous algorithms. Our approach subdivides the terrain into rectangular patches at different resolutions, and we present efficient algorithms for partitioning meshes into triangle strips, which are stitched together by some strips and commendably resolved the gaps between tiles. By combining carefully designed data structures with the use of GPU based programs, Out-of-core; multi-thread and view-dependent rendering technologies are used. We evaluate our technique by applying it to dynamic and real data sets.

Quad-tree based data processing

Dealing with huge amounts of data is often a difficult challenge [3]. Simplification algorithms take a terrain model as input and produce a simplified version of it, which provides a coarser representation of the same terrain, based on a smaller data set. Most methods are based on the iterated or simultaneous applications of local operators that simplify small portions of the mesh by reducing the number of vertices. The patch hierarchy is constructed top-down by subdividing each patch into 2×2 children patches, similar to restricted quad-tree. A restricted quad-tree is a structure that ensures all the levels of neighboring quads differ by not more than one [3]. As shown in figure 1, a generic structure, this can be stored and accessed quickly and flexibly. Considering rendering efficiency and

This article is an open access article under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0)

graphic engine, the size of tiles should be power of 2. Too large size will go beyond the capability of CPU, and too small size will increase the frequency of data exchange. Since the paging size of Intel CPU is 4K Byte, we recommend the terrain width is 32, and the texture width is 256, which improve the storing competence and decrease the remained memory.

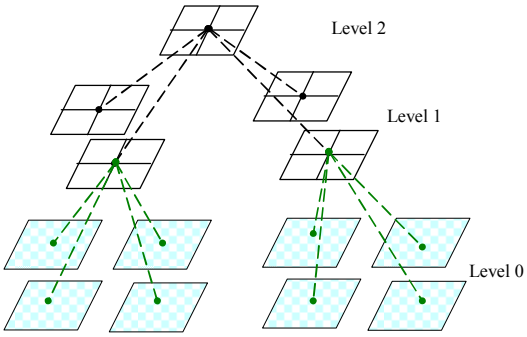


Fig. 1 A restricted quad-tree structure.

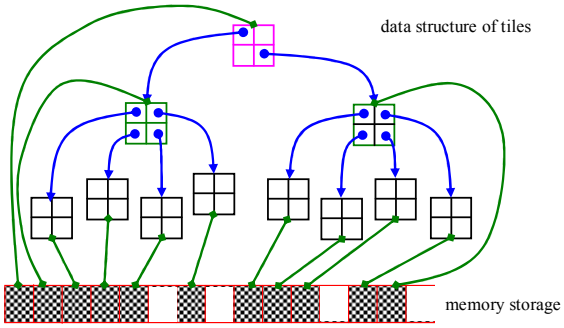


Fig. 2 Tiles stored in the memory.

Dynamic terrain rendering

Since the data include millions of triangle polygons and large scale of images, which make the models become much finer and complicated. The data is too massive to be loaded in the memory completely. **Multi-thread data loading.** For the usage of very large terrain models, it will be inevitable to use more than one processor in parallel [5]. Higher demands in memory, drawing speed and transfer efficiency appear. Multi-thread data loading was used, the main thread rendering all the data in memory; other threads accomplish data exchange, view culling, and terrain update. We set up third stage cache to lessen the time of data transfer, which includes disk storage, CPU and GPU memory. On one hand, independent parts of the landscape can then be updated independently. On the other hand, only multi-threading will ensure that the user-interaction and the update-process can be handled in parallel. Figure 2 is the data structure of tiles stored and exchanged in the memory, by which we can find parent and children directly.

In order to perform real-time rendering of massive data and considering current PC configuration, the internal memory only includes the data which need to be rendered. With the efficient algorithm of data schedule, these data can be updated real time. A separate I/O thread dynamically loads the tiles that are located inside the view frustum into main memory. Tiles that have left the pre-fetching region are removed from the tile tree and inserted into tile cache. That makes the best of file system driver development, which enhances the control of file access and the later guarantees the consistent of the data access and file arrangement [5]. Thus, the continuity of the data access can be guaranteed to give an excellent presentation depending on the viewer's perspective.

View-dependent culling and rendering. Maximum reduction of triangles rendered in every frame is the common goal is a key when roaming in large scale terrain [1-2]. The update thread traverses the scene in top-down manner and refines the tiles based on current parameters. Screen based error measure, expressed in pixel differences, is widely used in the literature because it relates to the overall visual quality directly. A well-known problem with this measure is that the higher patch doesn't take account of those pixel errors of patches of lower layers. We select layers and tiles using inversion of the camera parameters, which intersect terrain with a polygon area, as shown in figure 3. This square of tiles assures that the user can always look around at 360 degrees; all the tiles to load or to remove will be computed and labelled when update. Our tile management is thus quite similar to a classical paging system for terrains. However, the size of the square area is set according to the available main memory which makes it adaptive to the machine that is used for visualization.

The memory to be used for data exchange is tuned by an adaptive parameter, which represents the percentage of available memory that can be used after the non adaptive part of the environment is fetched. Knowing the amount of memory to be used, the square area is maintained by tracking the viewpoint by fetching/removing tiles to/from memory.

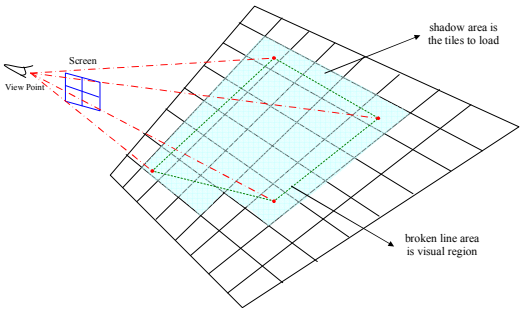


Fig. 3 View-dependent rendering.

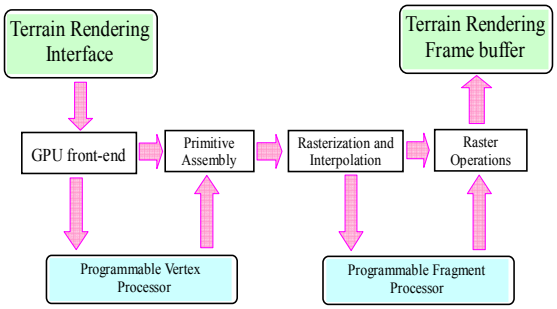


Fig. 4 GPU pipeline in terrain rendering.

GPU based acceleration. During runtime, the quad-tree of tiles is partially held and maintained in main memory, dependent on the movements of the viewer. In each update, tiles inside the view frustum are loaded and GPU based pipeline transform them to the frame buffer, as shown in figure 4. The rendering pipeline is mapped onto current graphics acceleration hardware such that the input to GPU is in the form of vertices [6]. These vertices then undergo transformation and per-vertex lighting. At this point in modern GPU pipelines a custom vertex shader program can be used to manipulate the 3D vertices prior to rasterization. Once transformed and lit, the vertices undergo clipping and rasterization resulting in fragments. A second custom shader program can then be run on each fragment before the final pixel values are output to the frame buffer for display.

The graphics pipeline is well suited to the rendering process because it allows the GPU to function as a stream processor since all vertices and fragments can be thought of as independent[6]. This allows all stages of the pipeline to be used simultaneously for different vertices or fragments as they work their way through the pipe. In addition to pipelining vertices and fragments, their independence allows graphics processors to use parallel processing units to process multiple vertices or fragments in a single stage of the pipeline at the same time [5]. Recent generations of GPU have allowed the design of algorithms that have a substantial part of their workload performed by the GPU. To reduce memory and bandwidth requirements, data compression employed, which can be decoded directly on the GPU.

Seamless Tiles Generation

Since tiles are loaded and rendered independently, the joints of vertex and texture between neighbouring tiles are key problems for large scale terrain roaming.

Connection of Meshes. Normalized meshes that represent the regular tiles and their stitching strips are generated at runtime. The patch hierarchy is used to guide the selection of the various levels of detail based on view-parameters. In each frame the patch hierarchy is traversed in a top-down manner to select a set of active tiles that form an appropriate level of detail. Since LOD representation of terrain meshes, different resolution of sample will occur between adjacent tiles, named T-connection, as shown in figure 5. To link all the tiles together, we render a mesh skirt down the surface around every tile. There are two methods to eliminate cracks caused by T-Connection. Filling method: That inserts new points in the low resolution tiles, and sets the value equal to corresponding points in the high resolution tiles. Then add new triangles in the mesh skirt, as shown in figure 5 (c). Omitting method, that ignores redundant points in the high resolution tiles, and then reconstructs triangles between neighbouring terrain meshes, as shown in figure 5 (d). We introduce the latter, because the resolution of mesh grid change acutely at the boundary of tiles, reduction of triangles will not reduce the quality; nay, it actually relieves the grid changes.

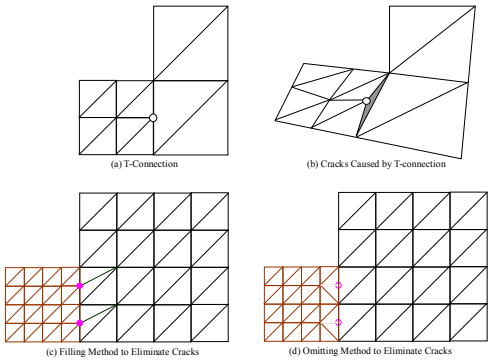


Fig. 5 Elimination of cracks.

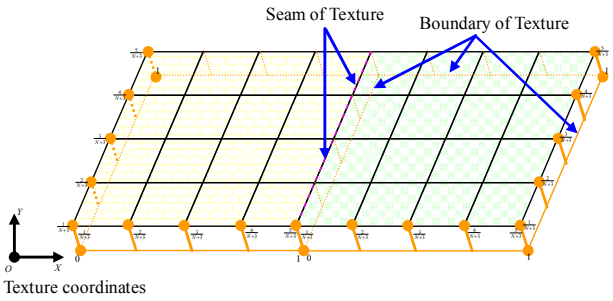
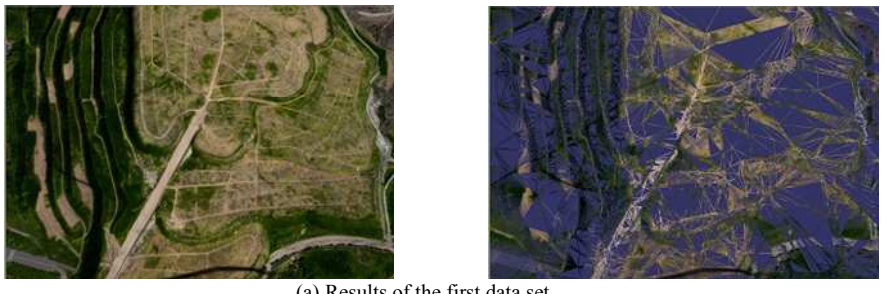


Fig. 6 Link textures of neighbouring tiles.

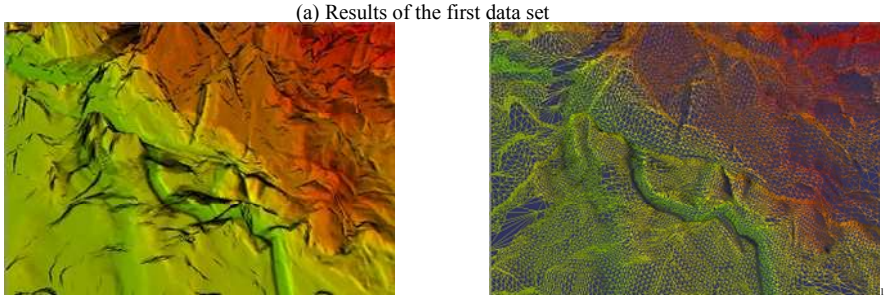
Connection of Textures. Since texture mapping was implemented in every tile interiorly, there is a seam obviously. Many of the current literatures did not resolve this matter commendably, which depress the quality of results [6]. Since OpenGL carries out texture mapping using bi-linear filter, we save a pixel of redundancy at every side of a block texture and used the same filter scheme and mesh skirt to relieve the seam between textures, as shown in figure 5. If the width of tile meshes is $N=2^n$ (n=1, 2 …), then the number of vertex in the tiles is (N+1)*(N+1), and we allocate a block memory of (N+3)*(N+3). Then a mesh skirt will be rendered under the terrain around a tile, as shown in figure 6. Since the real boundary of all tile textures will under the ground surface, and the seam of textures will lay the same position. That is to said, we put all of the tiles seamless together.

Results and Conclusions

We implemented above idea and method in MS visual studio C++ environment, based on low level graphic API, OpenGL, and the window size is in all cases 1024×768 pixels. We used a 2.66GHz Pentium Dual-Core CPU, with 2GB DDR2 of RAM, NVIDIA GeForce 9300M GS programmable graphic card with 256M RAM, and SATA 500G disk. We used two set of test data: The first is some area Quick Bird 0.6m high resolution image with size of 13108×14744 pixels, and digital elevation model grid of 2m resolution, with size of 3980×14744 vertex. The second is open Puget data, which is made up of 16385×16385 vertices at 10 meter horizontal and 0.1 meter vertical resolution, and the texture data is s made up of 16384×16384 pixels. Index of Quad-tree was set up based on file system, and figure 7 shows the print of a frame, including fill mode and wireframe mode.



(a) Results of the first data set



(b) Results of the second data set

Fig. 7 Experimental results of real data sets.

Based on above two data sets, we give a detail comparison on rendering performance under the same conditions with ROAM algorithm [1]. We sample the instant fps every second, with the same reference coordinates, the same roaming track, and the same moving speed. Figure 8 gives the fps changing graph of two methods with two data sets respectively. For our method, the average frame rate is 46.75 fps with the first data set, and 44.57 fps with the second data set; For ROAM algorithm, the average frame rate is 41.14 fps with the first data set, and 39.44 fps with the second data set. By a detailed analysis, we can see that ROAM takes a lot of time when processing vertex queue and mesh simplification, which reduces the rendering performance; contrarily, our method use the GPU for update and rendering step, which improve the output performance greatly.

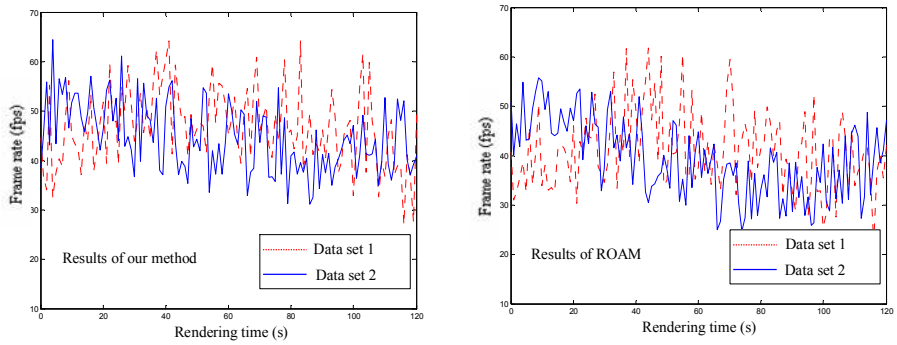


Fig. 8 Comparisons between our method and ROAM.

In figure 8 we can see that our method shows a good capability with large scale terrain, especially for real-time applications. The frame rates are between 50 to 110 fps, the nadir of frame rate is larger than 30 fps even though large data exchange occurs. Results from our analyses suggest that our method largely reduce the triangles number at every frame, and frame rates reached the real-time requirement of our travels. Although storage of the two data sets has biggish contrast, the changes of frame rate are unnoticeable. That is to say, our method is not sensitive to the data scale.

In this paper, we have presented a seamless rendering method for large scale height fields. We have demonstrated the effectiveness of our approach by comparing it to previous terrain simplification and rendering approaches. To future improve the scalability of the proposed terrain rendering pipeline we will investigate the possibility to use differently sized tiles in this approach. In this way we can adapt more flexibly to rendering load that is imposed by regions with different structures.

References

[1] M. Duchaineau. ROAMing terrain: real-time optimally adapting meshes. in Proceedings of the IEEE Visualization Conference. 1997. Phoenix, AZ, USA: IEEE Comp Soc, Los Alamitos, CA, United States.

[2] P. Renato and G. Enrico. Survey of semi-regular multiresolution models for interactive terrain rendering. Visual Computer, 2007. 23(8): p. 583-605.

[3] Y.Livny, Z. Kogan, and J. El-Sana. Seamless patches for GPU-based terrain rendering. Visual Comput, 2009, 25: 197-208, 2010.

[4] R. Kooima. Planetary-scale Terrain Composition. IEEE Trans Visualization and Computer Graphics, 15(5):719–733, 2009.

[5] M. Lambers and A. Kolb. GPU-Based Framework for Distributed Interactive 3D Visualization of Multimodal Remote Sensing Data. Int. IEEE Geoscience and Remote Sensing Symposium, 2009, 2009.

[6] Y. Livny, Z. Kogan and J. El-Sana. Seamless patches for GPU-based terrain rendering. Visual Computer, 2009. 25(3): p. 197-208.